Lisa Robinson

November 13, 2023

Foundations of Programming: Python

Assignment 05

https://github.com/Lisa-jr-batista/IntroToProg-Python-Mod05

Advanced Collections and Error Handling

Introduction

This week Arya introduced dictionaries, JSON files, structured error handling, and managing code files with tools like GitHub. Some key takeaways are that dictionaries are similar to lists but might be better when using more complex datasets with specific keys– they allow the user to more intuitively find specific elements rather than by row (as with a list). JSON files generally are more flexible than CSV and CSV can become less intuitive for the reader as the tables become more complex with many columns. Finally we spent time exploring error handling with try/except commands and if not/raise commands to predict and handle potential issues our program might face.

Writing the Program

I decided to start this week by copy and pasting the Assignment 5 starter python file. I felt after doing several labs and adapting my written code, mine was beginning to get messy so an option of a fresh start was really nice. I added in my info for the header shown in figure 1.

```
1  # ------------------------------------------------------------------------------------------
2  # Title: Assignment05
3  # Desc: This assignment demonstrates using dictionaries, files, and exception handling
4  # Change Log: (Who, When, What)
5  #   LRobinson,11/13/2023,Created Script
6  # ------------------------------------------------------------------------------------------
```

Figure 1 - Script header

I decided to give the starter code a run to see if I needed to troubleshoot any issues and right away got a "file not found" error. The "enrollments.csv" didn't exist yet on my computer. This was actually a great error to run into because I realized I couldn't remember the commands to create a file from scratch. I was able to quickly review module 2 and add in a csv write and f-string. This is shown in figure 2.

```
# Create Enrollments csv to be read later.
file = open(FILE_NAME, "w")
file.write(f'{student_first_name},{student_last_name},{course_name}')
file.close()
```

Figure 2 - Creating the needed csv file with appropriate formatting

After testing the code to ensure each part of the while loop worked correctly, I went in to add exceptions for error handling. I was a little confused about the first requirement: "The program provides structured error handling when the file is read into the list of dictionary rows." I decided that this was providing and error message for the scenario I described for figure 2– If the file does not exist. I went ahead and added a try/except command for "file not found" but when I went to test it, I realized I had already fixed this issue by adding in a "write" command earlier. When I changed the name of the file in the constant, my code automatically wrote it. Figure 3 shows the try/except error handling. Later I needed to change the write command ("w") to an append in order to add to the file and not truncate it with each run.

```
38  try:   # Addressing the file not found error
39      file = open(FILE_NAME, "r")
40      for row in file.readlines():
41          # Transform the data from the file
42          student_data = row.split(',')
43          student_data = [student_data[0], student_data[1], student_data[2].strip()]
44          # Load it into our collection (list of lists)
45          students.append(student_data)
46      file.close()
47  except FileNotFoundError as e:
48      print("Text file must exist before running this script!\n")
49      print("-- Technical Error Message -- ")
50      print(e, e.__doc__, type(e), sep='\n')
51  finally:
52      if file.closed == False:
```

Figure 3 - Try/except error handling for "file not found."

I moved into the input error messages and started with a try/except command and added in "if not/ raise" commands for both the first and last name inputs. I started with the "if not student_first_name.isalpha():" command to raise the value error so ONLY letters could be entered into my csv. However, I ran into a problem when I wanted to enter in a hyphenated last name. The error was raised because a hyphen is a special character and not alphabetic. With some outside research I was able to change the command to

what you see in figure 4. I dropped the “if not” for “if” and added the isnumeric command. This allowed me to add in the hyphenated last name.

```
student_last_name = input("Enter the student's last name: ")
if student_last_name.isnumeric():
    raise ValueError("The last name should only contain letters. ")
```

Figure 4 - changing the if statement to allow special characters.

Running the code from here worked, but I still had some issues. When I tried to trip up the system by typing in a first name like “6of9” it still allowed it with no error because it contains SOME letters. My code would only raise the value error if I tried to input something that was entirely numbers. Not ideal. This led me down a path that eventually ended with discovering the “ if any” command on Stack Overflow. I was able to ask my code to check that each character in my input was a number, and if it was, the “if any” command would output “false” , raising my error message. Figure 5 shows that new code.

```
student_last_name = input("Enter the student's last name: ")
if any(char.isdigit() for char in student_last_name):
    raise ValueError("The last name should only contain letters. ")
```

Figure 5 - “if any” command to check input for any digit and raise the value error.

After spending time making sure I had the correct error handling message for the input, I also added in a try/except command for menu option 3 when the information would be saved to the csv. When we learned this in class and through the notes, we worked with JSON files. Since the assignment did not require this, I left mine in CSV. So I went in to change the error message to ensure the data is a valid csv format. This is shown in figure 6. I was unable to trip this error message, however, so I think I still need to learn in this area.

```
93      elif menu_choice == "3":
94          try:  # The program provides structured error handling when the dictionary rows are written to the file.
95              file = open(FILE_NAME, "a")
96              for student in students:
97                  csv_data = f"{student[0]},{student[1]},{student[2]}\n"
98                  file.write(csv_data)
99              file.close()
100             print("The following data was saved to file!")
101             for student in students:
102                 print(f"Student {student[0]} {student[1]} is enrolled in {student[2]}")
103             continue
104         except TypeError as e:
105             print("Please check that the data is a valid csv format\n")
106             print("-- Technical Error Message -- ")
107             print(e, e.__doc__, type(e), sep='\n')
108         except Exception as e:
109             print("-- Technical Error Message -- ")
110             print("Built-In Python error info: ")
111             print(e, e.__doc__, type(e), sep='\n')
112         finally:
113             if file.closed == False:
114                 file.close()
```

Figure 6 - Error handling when adding rows to the file.

My final step was testing my code in Terminal. I was able to pull up the correct directory and run my code with Python3. Figure 7 shows the code running in Terminal, and after making changes to the "write" command, I was able to hold onto the previous student entries I made and add more without deleting what already existed.

```
What would you like to do: 3
The following data was saved to file!
Student Lisa Batista is enrolled in Python 100
Student Ricardo Batista is enrolled in Python 100
Student Hero Robinson-Batista is enrolled in Python 100
Student Vic Vu is enrolled in Python 100

---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------

What would you like to do:
```

Figure 7 - Running in Terminal

Summary

I really enjoyed learning about a new file system with JSON and I plan to continue to explore how to best utilize it with future more complicated datasets. I did not need it for Assignment 05 but I want to try to use it more in the future so I can get more experience with the formatting requirements. I am proud I was able to explore new ways to problem solve with my error handling. It would have been easy to accept not being able to use a hyphenated name as an input, but I felt that understanding how to change the code to allow for it but NOT allow for numbers was important. Going through the trial and error process with that code was fun and I like that it felt somewhat low-stakes. That allowed me to keep trying and researching until I found an answer that worked. Where I still want to learn and grow is better understanding dictionaries and how they work. I still feel shaky with formatting and fully understanding the difference or why a dictionary might be preferred over lists.